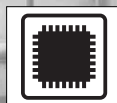
基礎から学ぶ Verilog HDL & FPGA 設計

第5回

ステート・マシンの設計

中野浩嗣，伊藤靖朗

本連載は，さまざまな回路をVerilog HDLで設計していき，最終的には小型のCPUを実現することを狙いとしている．今回は，CPUの状態を制御するステート・マシンを作る．ステート・マシンを順序回路として設計し，シミュレーションとFPGAを用いた動作確認を行う．（筆者）

● CPUの基本動作と状態遷移図

基本的には，CPUは次の二つの動作を繰り返します．

●命令フェッチ

メモリに格納されている機械語命令を取り出す（フェッチする）．

●命令実行

取り出した機械語命令を実行する．

さらに，動作を行っていない**待機**状態を加えると，CPUの基本動作は，**図1**のように表すことができます．このような図を**状態遷移図**と呼びます．

図1の状態遷移図の矢印は，状態遷移規則を表しています．遷移要求があるたびに遷移が行われます．状態遷移規則には，無印のものと，**動作開始**と**動作終了**のラベルの付いたものがあります．一つの状態から二つ以上の外向きの矢印が出ている場合，外部からの合図によって遷移先の状態が決まります．例えば，待機状態で動作開始の合図があると，命令フェッチ状態に遷移します．待機状態で合図がない場合は，待機状態に遷移，つまり状態が変わりません．命令フェッチ状態からは，命令実行状態に遷移し，命令実行状態で合図がない場合は，命令フェッチ状態に戻ります．命令実行状態で動作終了の合図があると，待機状態に遷移します．従って，動作終了の合図がない間，命令フェッチと命令実行を交互に繰り返します．

状態遷移図により定義された状態遷移を実現するものがステート・マシンです（**図2**）．このステート・マシンでは，遷移要求，および動作開始と動作終了の合図が入力され，現在の状態を出力します． 遷移要求があるたびに，**図1**の状態遷移図に従って状態遷移を行います．

● ステート・マシンの設計

現在の状態を保持するのにフリップフロップを用いれば，ステート・マシンは順序回路として設計することができます．実際にCPUで用いるステート・マシンを設計しましょう．本連載で設計するCPUの状態遷移はもう少し複雑で

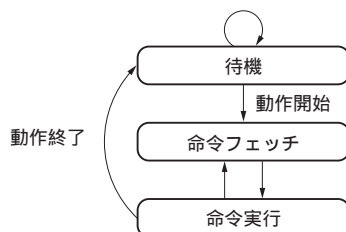



図1 CPUの基本動作

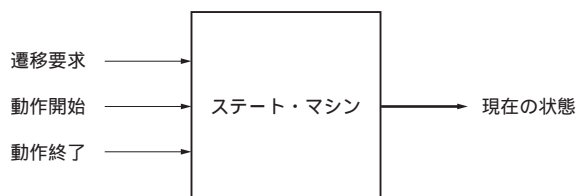



図2 ステート・マシン

KeyWord 命令フェッチ，命令実行，状態遷移図，動作開始，待機状態，動作終了

す．**図3**はその状態遷移図です．

五つの状態を持ち，IDLEが待機状態，FETCHAとFETCHBが命令フェッチのための状態，EXECAとEXECBが命令実行のための状態です．命令フェッチと命令実行のために，それぞれ2クロック・サイクルが必要なため，二つの状態を割り当てています．**図4**はこの状態遷移図を実現するステート・マシンです．

このステート・マシンは，後で順序回路に実装するのを想定して設計します．ステート・マシンは，clk，reset，run，cont，haltの五つの入力を持ちます．出力は現在の状態csです．

基本的に，クロックclkの立ち上がりで状態遷移が行われます．リセットresetが0になると，clkの立ち上がりに関係なくIDLEに非同期に遷移します．状態がIDLEのときrunが1ならば，FETCHAに遷移します．その後は基本的に，FETCHA→FETCHB→EXECA→FETCHA→というように三つの状態FETCHA，FETCHB，EXECAを順に遷移し，命令フェッチと命令実行を繰り返します．

状態がEXECAのとき，haltが1なら，IDLEに遷移します．これはCPUの動作の終了を意味します．

状態がEXECAのとき，contが1なら，EXECBに遷移します．これは，EXECAの1クロック・サイクルだけでは命令実行が完了せず，2クロック・サイクル必要な場合に対応します．EXECBに遷移した後，FETCHAに戻ります．

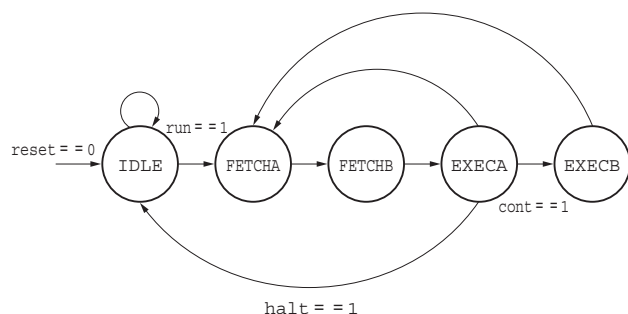



図3　CPUの状態遷移図

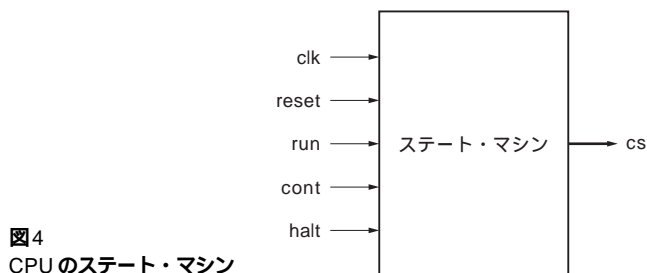



図4
CPUのステート・マシン

● ステート・マシンのVerilog HDL記述

ステート・マシンを順序回路で実装するために，レジスタ（フリップフロップ）を用いて現在の状態を記憶します．**図3**のステート・マシンは五つの状態を持つので，3ビットのレジスタ（つまり3個のフリップフロップ）を用います．3ビットあれば，2進数表現で000から111までの八つ（$2^3=8$）の状態を区別することができるので十分です．

2ビットでは最大4（$=2^2$）状態なので，5状態を区別するには不足します．

リスト1は，この方法によりステート・マシンを実現するVerilog HDL記述です．11行目のreg文で，3ビットのレジスタcsを宣言します．このレジスタを用いてステート・マシンの状態を保持することにします．

1行目～5行目では，define文を用いて，状態名と値を対応づけています．

13行目から始まるalways文でcsの値を決定しています．

14行目は，resetが0のときcsの値がIDLEになる非同期リセットを定義しています．

16行目から始まるcase文では，resetが1でclkの立ち上がりが発生したときの遷移先を定義しています．csの値がIDLEの場合，17行目のif文が実行され，runが1のときcsにFETCHAつまり，3'b001が代入されます．runが0のときは代入が行われないので，csはIDLEのままです．csがFETCHAのときはFETCHBに，FETCHBの場

リスト1　ステート・マシンのVerilog HDL記述（state.v）

```
1    'define IDLE 3'b000
2    'define FETCHA 3'b001
3    'define FETCHB 3'b010
4    'define EXECA 3'b011
5    'define EXECB 3'b100
6
7    module state(clk,reset,run,cont,halt,cs);
8
9      input clk, reset, run, cont, halt;
10     output [2:0] cs;
11     reg    [2:0] cs;
12
13     always @(posedge clk or negedge reset)
14       if(!reset) cs <= 'IDLE;
15       else
16         case(cs)
17         'IDLE: if(run) cs <= 'FETCHA;
18         'FETCHA: cs <= 'FETCHB;
19         'FETCHB: cs <= 'EXECA;
20         'EXECA: if(halt) cs <= 'IDLE;
21                else if(cont) cs <= 'EXECB;
22                else cs <= 'FETCHA;
23         'EXECB: cs <= 'FETCHA;
24         default: cs <= 3'bxxx;
25       endcase
26
27   endmodule
```

合はEXECAがcsに代入されます．csがEXECAのとき，haltが1ならばIDLEがcsに代入されます．contが1ならば，EXECBが代入されます．いずれでもない場合，FETCHAが代入されます．状態がEXECBのとき，次の状態は常にFETCHAです．

24行目のdefafult文は，csに不定値3'bxxxを代入しています．これは，csが定義した五つの値以外の値を取ることはありえないので，このような場合csにどのような値を代入してもかまわないことを意味しています．これにより，五つの値以外を取る場合については無視して回路を生成できるので，設計ツールはよりコンパクトな回路を生成することが期待できます．また，default文を省略したとしても，組み合わせ回路の場合のように，非同期ラッチを生成することはありません．もし省略したとしても，五つの値以外の場合は，csの値を変更しないような回路となります．しかし，default文がある場合に比べて，より複雑な回路になる可能性があるので省略しないほうがよいでしょう．

● ステート・マシンのシミュレーション

リスト1のステート・マシンの動作をシミュレーションで確認してみましょう．**リスト2**はステート・マシンのテスト・ベンチです．テスト・ベンチの基本的な構造は，前回作成したカウンタのテスト・ベンチとほぼ同じです．

4行目でステート・マシンへの五つの入力をレジスタ変数として宣言します．

5行目でステート・マシンの出力をネット変数として宣言します．

7行目でステート・マシンをインスタンス化します．

9行目から13行目で50nsごとに値が反転するクロックclkを定義します．15行目以降でreset，run，halt，contの四つの値の変化を定義します．

図5は**リスト2**のシミュレーション結果です．シミュレーション開始時はresetが0なので，状態csは3'b000，つまりIDLEです．時刻100nsから200nsの間，runが1なので，時刻150nsのclkの立ち上がりで状態csは3'b001，つまりFETCHAに遷移します．

続けて，状態csが3'b010(FETCHB)，3'b011(EXECA)と遷移します．時刻400nsから500nsの間，contが1なので，状態は3'b100(EXECB)に遷移します．その後，状態は3'b001に戻り，以下同様に状態遷移を繰り返します．時刻1150nsにおいて，状態csは3'b011(EXECA)であり，haltが1なので，次の状態は3'b000

リスト2　ステート・マシンのテスト・ベンチのVerilog HDL記述(state_tb.v)

```
1     'timescale 1ns / 1ps
2     module state_tb;
3
4      reg clk,reset,run,halt,cont;
5      wire [2:0] cs;
6
7     state state0(.clk(clk),.reset(reset),.run(run),
                   .cont(cont),.halt(halt),.cs(cs));
8
9     initial begin
10    clk = 0;
11    forever
12       #50 clk = ~clk;
13    end
14
15    initial begin
16      reset = 0; run = 0; halt = 0; cont = 0;
17      #100 reset = 1; run = 1;
18      #100 run = 0;
19      #200 cont = 1;
20      #100 cont = 0;
21      #600 halt = 1;
22      #100 halt = 0;
23     end
24
25    endmodule
```

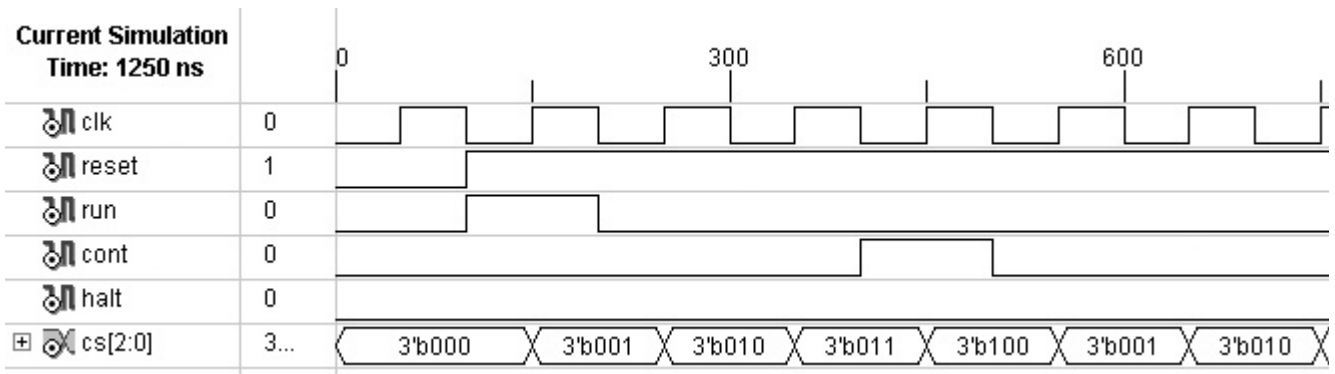



図5　ステート・マシンのシミュレーション波形

(IDLE)になります．以上のことから正しく動作していることが確認できます．

● ステート・マシンの FPGA ボードによる動作確認

最後に，米国Xilinx社の「Spartan-3E スタータ・キット」または「Spartan-3A スタータ・キット」のFPGA ボードを用いて，ステート・マシンの動作を確認してみましょう．

リスト3　トップ・モジュールのVerilog　HDL記述(state_top.v)

```
1    'define IDLE 3'b000
2    'define FETCHA 3'b001
3    'define FETCHB 3'b010
4    'define EXECA 3'b011
5    'define EXECB 3'b100
6
7    module state_top(BTN_EAST, BTN_SOUTH, SW, LED);
8     input BTN_EAST, BTN_SOUTH;
9     input [2:0] SW;
10    output [4:0] LED;
11    wire [2:0] cs;
12
13   state state0(.clk(BTN_EAST),.reset(~BTN_SOUTH),
            .run(SW[2]),.cont(SW[1]),.halt(SW[0]),
             .cs(cs));
14
15    assign LED[4] = (cs == 'IDLE);
16    assign LED[3] = (cs == 'FETCHA);
17    assign LED[2] = (cs == 'FETCHB);
18    assign LED[1] = (cs == 'EXECA);
19    assign LED[0] = (cs == 'EXECB);
20
21   endmodule
```

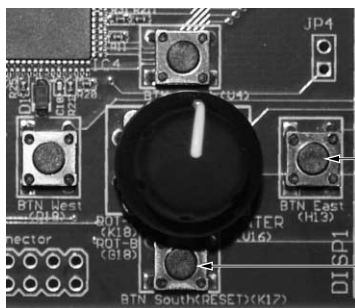



写真1　プッシュ・スイッチ
右側のスイッチにclk，下側のスイッチにresetを接続する．

具体的には，ステート・マシンの入力をプッシュ・スイッチとスライド・スイッチに割り当て，状態遷移の様子をLEDで表示するようにします．

前回のカウンタの動作確認を行ったときと同様に，FPGAボードとのインターフェースとなるモジュールを作成します．

リスト3はそのモジュールstate_topです．入力は，1ビットのBTN_EASTとBTN_SOUTH，および3ビットのSWです．出力は5ビットのLEDです．これらのポートはユーザ制約ファイル(UCF)によるピン割り当てにより，FPGAボード上のスイッチやLEDと接続します．

BTN_EASTとBTN_SOUTHは，右側と下側に位置するプッシュ・スイッチ(**写真1**)に接続します．3ビットのSWは，FPGA ボード上の4個あるスライド・スイッチ(**写真2**)のうち，右側の3個に接続します．5ビットのLEDは，FPGAボード上の8個のLEDのうち，右側の5個に接続します．これらの接続関係を定義するユーザ制約ファイルが**リスト4**(Spartan-3E スタータ・キット用)と**リスト5**(Spartan-3A スタータ・キット用)です．ピン割り当てに

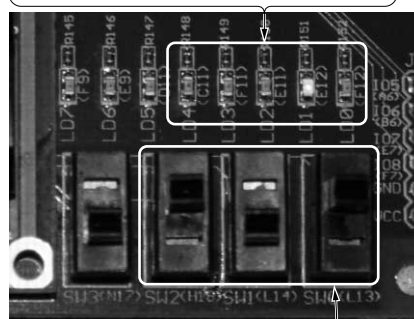



写真2　スライド・スイッチとLED
右から3番目のスライド・スイッチにrun，右から2番目のスイッチにcont，1番右側 のスイッチにhaltを接続する．また，LEDの右側5個を用いてステート・マシンの 現在の状態を表す．

**リスト4
トップ・モジュールのユーザ制約ファイル(state_top.ucf，Spartan-3E スタータ・キット用)**

```
1     # PUSH SWITCH
2     NET "BTN_EAST" LOC = "H13" | IOSTANDARD = LVTTL | PULLDOWN ;
3     NET "BTN_SOUTH" LOC = "K17" | IOSTANDARD = LVTTL |PULLDOWN ;
4
5     # LED
6     NET "LED<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
7     NET "LED<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
8     NET "LED<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
9     NET "LED<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
10    NET "LED<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
11
12    # SLIDE SWITCH
13    NET "SW<2>" LOC = "H18" | IOSTANDARD = LVTTL | PULLUP ;
14    NET "SW<1>" LOC = "L14" | IOSTANDARD = LVTTL | PULLUP ;
15    NET "SW<0>" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ;
```

リスト5
トップ・モジュールのユーザ制約ファイル（state_top.ucf，Spartan-3A スタータ・キット用）

```
1     # PUSH SWITCH
2     NET "BTN_EAST" LOC = "T16" | IOSTANDARD = LVTTL | PULLDOWN ;
3     NET "BTN_SOUTH" LOC = "T15" | IOSTANDARD = LVTTL |PULLDOWN ;
4
5     # LED
6     NET "LED<4>" LOC = "V19" | IOSTANDARD = LVTTL | SLEW =QUIETIO | DRIVE = 4 ;
7     NET "LED<3>" LOC = "U19" | IOSTANDARD = LVTTL | SLEW =QUIETIO | DRIVE = 4 ;
8     NET "LED<2>" LOC = "U20" | IOSTANDARD = LVTTL | SLEW = QUIETIO | DRIVE = 4 ;
9     NET "LED<1>" LOC = "T19" | IOSTANDARD = LVTTL | SLEW =QUIETIO | DRIVE = 4 ;
10    NET "LED<0>" LOC = "R20" | IOSTANDARD = LVTTL | SLEW = QUIETIO | DRIVE = 4 ;
11
12    # SLIDE SWITCH
13    NET "SW<2>" LOC = "U8" | IOSTANDARD = LVTTL | PULLUP ;
14    NET "SW<1>" LOC = "U10" | IOSTANDARD = LVTTL | PULLUP ;
15    NET "SW<0>" LOC = "V8" | IOSTANDARD = LVTTL | PULLUP ;
```

ついての詳細はユーザ・マニュアル[(1)，(2)]を参照してください．

リスト3の13行目で，モジュールstateのインスタンス化を行います．この時，入力ポートclkとBTN_EASTを接続します．従って，右側のプッシュ・スイッチを押すと，clkが1になります．また，入力ポートresetには，BTN_SOUTHの論理否定~BTN_SOUTHを接続します．従って，下側のプッシュ・スイッチを押したときには，resetが0になり，非同期リセットが行われます．

入力ポートSW[2]にrun，SW[1]にcont，SW[0]にhaltを接続します．こららの値は，三つのスライド・スイッチの位置により決まり，上の時は1，下の時は0です．モジュールstateが出力する現在の状態csは，11行目で宣言したネットcsにそのまま接続します．

連載第1回（本誌2007年4月号，pp.105-114）で説明した手順でビットストリーム・ファイルstate_top.bitを作成し，このビットストリーム・ファイルをFPGAにダウンロードします．ビットストリーム・ファイルを作成する際に，state_top.vがトップ・モジュールであり，**リスト4**，または**リスト5**のユーザ制約ファイルがプロジェクトに追加されていることを確認します．プロジェクトに設定できるユーザ定義ファイルは一つだけなので，もしほかのユーザ制約ファイルが設定されている場合，これをいったん削除して，必要なユーザ制約ファイルを追加します．エラーがなければ，ビットストリーム・ファイルstate_top.bitが生成されるので，これをFPGAにダウンロードします．

右側のプッシュ・スイッチを押すたびに，clkの立ち上がりが発生し，状態遷移が起こります．最初は，状態がIDLEですが，右から3番目のスライド・スイッチを上にするとrunが1になります．このとき，右側のプッシュ・スイッチを押すとFETCHAに遷移します．**図4**の状態遷移図の通り正しく状態遷移することが，LEDの点灯により確認できるはずです．

次回は数式の評価に用いるスタックを設計します．

参考・引用*文献

（1）Xilinx；Spartan-3E スタータ・キット ボード・ユーザー・ガイド，http://japan.xilinx.com/bvdocs/userguides/ug230.pdf
（2）Xilinx；Spartan-3A スタータ・キット ボード・ユーザー・ガイド，http://japan.xilinx.com/bvdocs/userguides/ug330.pdf

なかの・こうじ
いとう・やすあき
広島大学大学院工学研究科

<筆者プロフィール>

中野浩嗣．1992年大阪大学大学院博士後期課程修了．工学博士．一つの民間企業，二つの大学を経て，2003年より，広島大学教授．
伊藤靖朗．2003年北陸先端科学技術大学院大学博士前期課程修了．現在，広島大学助教．